

# Hardware and software aspects of screen handling on the VZ-200/300

Concluding with coverage of the software interface in the VZ and the MC6847 VDG, looking at the standard screen modes.

**Part 2**  
**Bob Kitch**

IT IS NOW OPPORTUNE to briefly discuss the software interface in the VZ and the VDG. I will only discuss the standard screen modes used on the VZ — not the additional modes mentioned in Part 1.

## Lo-res/Text/Mode (0).

In the lo-res mode the screen is formatted into 16 lines down the usable window with each line containing 32 characters. Thereby providing 512 addressable characters on the screen. A quick calculation (or look at Table 1) will show that each character is composed of 8 by 12 pixels (or dots). Furthermore, each character is 'described' in a single byte in the video RAM section of memory. The upper left-hand character on the screen is memory mapped onto address 7000H (28672D), and the lower right-hand character is mapped onto 7000H + 1FFH (29183D). A memory map for the lo-res screen is given in Figure 5.

A formula is often used to calculate the address of a particular character on the screen. Let AA be the position of the character ACROSS the line (which ranges from 0 to 31) and let AD be the line number DOWN the screen (ranges from 0 to 15). i.e: working in the SE quadrant of an X-Y axis system. The relationship between (AA,AD) and the address in RAM is —

$$\text{MAPPED ADDRESS} = \text{START ADDRESS} + (32 * \text{AD} + \text{AA}) \text{ or}$$

$$\text{Addr} = 28672 + (32 * \text{AD} + \text{AA})$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	P		0	Q	R	S	T	U	V	W	X	Y	Z	[	]
1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
2	R	S	T	U	V	W	X	Y	Z	[	]	^	_	`	a	b
3	C	S	#	3	D	E	F	G	H	I	J	K	L	M	N	O
4	D	T	\$	4	U	V	W	X	Y	Z	[	]	^	_	`	a
5	E	U	%	5	F	G	H	I	J	K	L	M	N	O	P	Q
6	F	V	&	6	G	H	I	J	K	L	M	N	O	P	Q	R
7	G	W	'	7	H	I	J	K	L	M	N	O	P	Q	R	S
8	H	X	(	8	I	J	K	L	M	N	O	P	Q	R	S	T
9	I	Y	)	9	J	K	L	M	N	O	P	Q	R	S	T	U
A	J	Z	*	:	K	L	M	N	O	P	Q	R	S	T	U	V
B	K	[	+	;	L	M	N	O	P	Q	R	S	T	U	V	W
C	L	\	,	<	M	N	O	P	Q	R	S	T	U	V	W	X
D	M	]	-	=	N	O	P	Q	R	S	T	U	V	W	X	Y
E	N	^	.	>	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	O	/	?	@	P	Q	R	S	T	U	V	W	X	Y	Z	[

**TABLE 2.**

**Alphanumeric and Semigraphic 4 character set for the VZ-200 and VZ-300 held in MC6847 on-chip ROM. (Users — note errors in shape table held in VZ ROM for inverse J, X, 3 and 5).**

G Y B R BF CN M O  
-----  
(COLOURS)



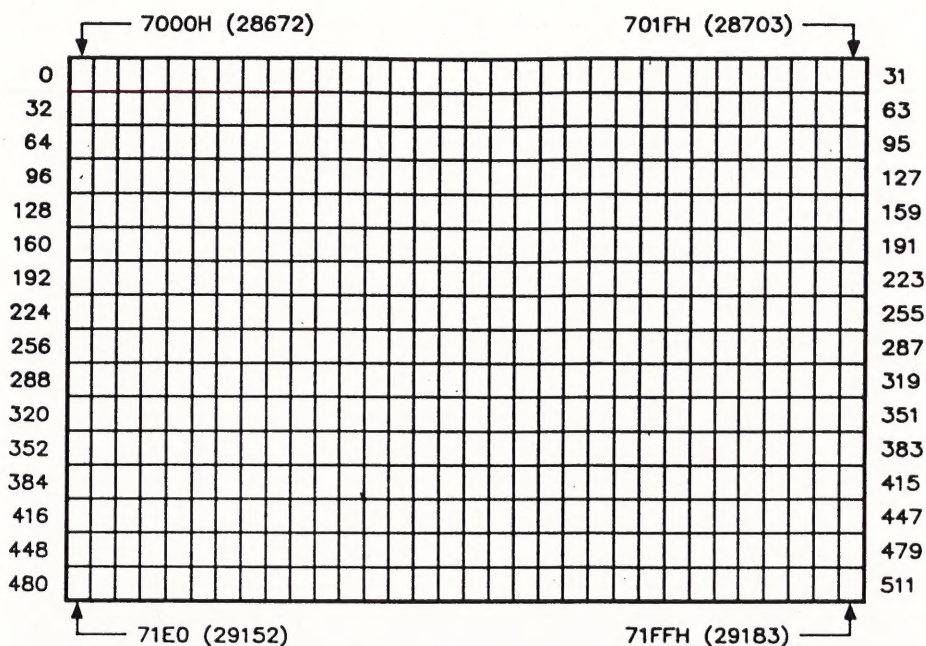


Figure 5. Screen addressing for MODE (0) or lo-res displays on VZ computers. This mode corresponds to Alphanumeric and Semigraphic 4 on VDG and is 32 by 16 characters in size. Each character is byte-mapped as indicated.

being selectable from green or orange. Remember that the Inverse mode of the MC6847 is set by bit 6 of the data value contained in video RAM. (see also Figures 1 and 6).

An understanding of this involves looking at individual bits within the bytes and also looking at how these bits can control and reset certain control lines on the VDG (as outlined in Part 1).

In text mode there are 64 characters in each of the Normal (0-63) and Inverse (64-127) sets. This implies that a 6-bit code is used to encode the character shape and that bit 6 determines whether Normal or Inverse.

For example:—

b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	1	0	1	Binary = 37D or '% normal.
0	1	1	0	0	1	0	1	Binary = 101D or '% inverse.

Note the way that bit 6 determines normal/inverse. Also note that bit 7 does not change. The most significant bit (MSB) is used to indicate text character to the on-chip ROM.

In summary, for the character source, a 6-bit ASCII code is used to call the element from the on-chip ROM, the seventh bit indicates normal or inverse illumination, and the eighth bit is held low to indicate Alphanumeric mode.

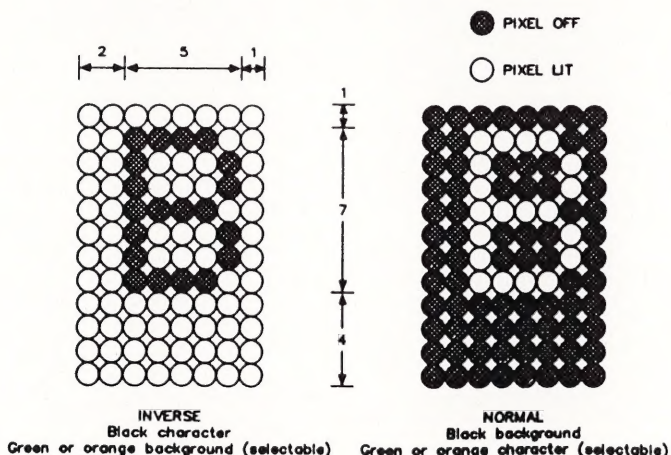


Figure 6. Format of Alphanumeric Mode — Internal on MC6847. Each character is 12 by 8 pixels and each screen is 32 by 16 characters. A 6-bit ASCII code specifies the character from an on-chip ROM.

b7	b6	b5	b4	b3	b2	b1	b0
alpa *A/S	inv INV	6-bit ASCII					

In graphics mode the Semigraphics 4 mode of the VDG is used. The 8 by 12 pixel character is divided into four 'rectangular' quadrants of size 4 by 6. The quadrants are 'pseudo-addressable' by selecting the correct area as shown on Figure 7.

In Semigraphics mode, a more comprehensive form of encoding is used. The character codes extend from 128 to 255, implying that the MSB (or bit 7) is set to 1 (or high) to indicate that a graphics character is encoded in the byte. The graphic block character contains 16 discrete patterns involving 'switching' on or off the four quadrants. The four low-order bits handle a quadrant a piece (refer Figure 7). Additionally one-of-eight illumination colours is encoded in the next three bits (bits 6 to 4).

For example:—

b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	1	0	1	Binary = 217D or  cyan
0	1	1	0	0	1	0	1	Binary = 145D or  yellow

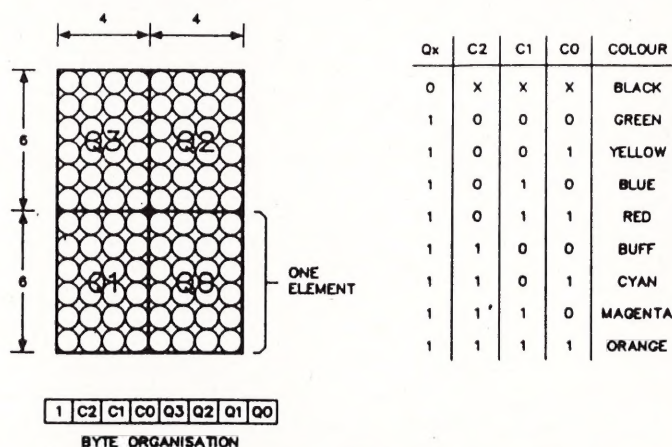
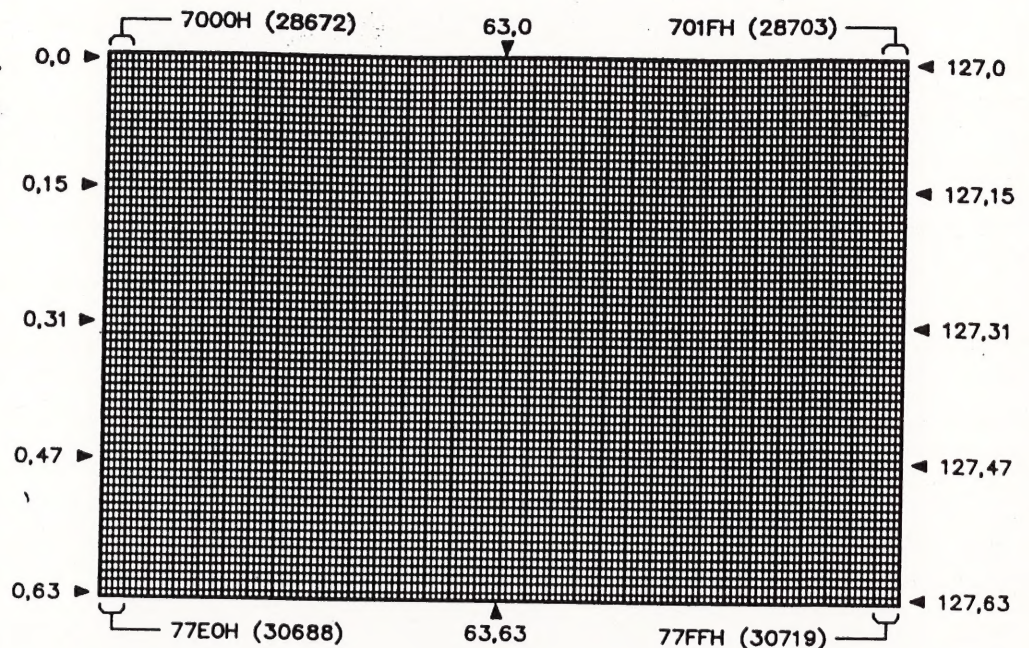


Figure 7. Format of Semigraphic 4 Mode on MC6847. Each character is 12 by 8 pixels but elements or quadrants can be individually illuminated giving a screen resolution of 64 by 32 elements in up to eight colours.



Figure 8. Screen Addressing for MODE (1) or hi-res displays on the VZ computers. This mode corresponds to Colour Graphics 2 on the VDG and is 128 by 64 elements in size. Each element is mapped with two bits.



In summary, for Semigraphics mode it can be seen that each of the four least significant bits controls one of the quadrants, whilst the next three bits determine the colour of the illumination. The most significant bit is set high to indicate a graphics block is encoded.

b7	b6	b5	b4	b3	b2	b1	b0
graphic *A/S				G3	G2	G1	G0

In this mode, although the screen is formatted into 32 by 16 graphics blocks, in fact the quadrant resolution is actually 64 by 32 and with all of the eight colours available. This may be thought of as an intermediate resolution display mode.

Thus it can be seen that Alphanumerics in either Normal or Inverse style and Semigraphics blocks of up to eight colours can be individually set on the lo-res screen by byte mapping. Different forms of encoding the necessary information are used in each case. These features combine to make MODE(0) quite a powerful display despite its lack of resolution.

### Hi-res/Graphics/Mode(1)

In hi-res or MODE(1), the screen has 128 by 64 elements individually addressable. This corresponds to 8192 elements and with only 2K of video RAM available, then some sort of trade-off in features over lo-res must ensue. In hi-res, each element is 2 by 3 pixels in size and is (noticeably) rectangular in shape. Video RAM addressing extends from 7000H (28672D) to 71FFH (30719D) — 2048 bytes as shown in Figure 8.

This mode corresponds to Colour Graphics Two (CG2) on the VDG chip. Each byte addresses four consecutive elements across the screen. Each element may be one-of-four colours (selected from either of the two colour sets). Note the trade-off in colours and the different way in which elements are addressed on the screen — such that MODE(0) and MODE(1) screens cannot be mixed.

There are a couple of ways in which each element may be illuminated.

The simplest (and slowest) way is by using the BASIC commands of SET and RESET. These commands alter two bits of the appropriate byte in the video RAM area. The processing is very slow because of this limitation and the fact that

it is done through the BASIC interpreter. Listing 1 provides a simple illustration of this method. The program fills the entire screen with hi-res elements according to the COLOR command. The use of integer index variables speeds up the program a little.

LISTING 1

```

10 '***SNAIL GRAPHICS DEMO***
20 '*** HI-RES ***
30 '*** VERSION 1.2 ***
40 '*** R.B.K. 22/5/86 ***
50 '*** EXECUTION TIME 43.7 SECS.
100 'SET TO HI-RES
120 MODE(1)
130 COLOR 3,0
140 SOUND 10,1
200 FOR VX=0 TO 63
210 FOR HX=0 TO 127
220 SET (HX,VX)
230 NEXT HX
240 NEXT VX
250 SOUND 10,1
260 STOP
270 END

```

LISTING 2

```

10 '***SNAIL GRAPHICS DEMO***
20 '*** HI-RES ***
30 '*** VERSION 2.3 ***
40 '*** R.B.K. 22/5/86 ***
50 '*** EXECUTION TIME 0.3 SECS.
100 'SET TO HI-RES
120 MODE(1)
130 COLOR ,0
140 VX=170: SOUND 10,1
200 FOR IX=28672 TO 30719
210 POKE IX,VX
220 NEXT IX
250 SOUND 10,1
260 STOP
270 END

```

LISTING 3

```

10 '***NEAR-LIGHT-SPEED GRAPHICS DEMO***
20 '*** HI-RES ***
30 '*** VERSION 1.2 ***
40 '*** R.B.K. 22/5/86 ***
50 '*** EXECUTION TIME 0.5 SECS.
100 '***LOAD BLOCK MOVE MACHINE CODE.***
110 FOR IX=-28687 TO -28674
120 READ AX: POKE IX,AX
130 NEXT
140 DATA 33,0,112,17,1,112,1,255,7,54,170,237,176,201
200 '***INITIALIZE USR() TO ADDRESS $FF1H OR -28687D.***
210 POKE 30842,241: POKE 30863,143
300 '***SET TO HI-RES.***
310 MODE(1)
320 COLOR ,0
330 SOUND 10,1
340 X=USR(0)
350 SOUND 10,1: SOUND 0,9
360 COLOR ,1
370 SOUND 10,1: SOUND 0,9
380 STOP
390 END

```



**LISTING 4**

```

10 *****
20 *** 2000 VZ SCREENS ***
30 *** VERSION 1.2 ***
40 *** R.B.K. 18/5/86 ***
50 *****
60
100 ***FIND TOP OF MEMORY.
110 HI=PEEK(30898);LI=PEEK(30897);***PRESERVE TOM POINTERS.
120 TH=HI*256+LI-20 ***RESERVE TOP 20 BYTES.
130 MS=INT(TH/256);LS=TH-MS*256
140 POKE 30898,MS;POKE 30897,LS
150
200 ***SET UP LOADING OF USR() ROUTINE.
210 TH=TH+1 ***NEXT ADDR IN RESERVED MEM.
220 MS=INT(TH/256);LS=TH-MS*256
230 POKE 30863,MS;POKE 30862,LS
240 AD=TH+10 ***ADDR. FOR CHARACTER BYTE.
250 IF TH>32767 THEN TH=TH-65536 ***CONVERT TO SIGNED INTEGER.
260 IF AD>32767 THEN AD=AD-65536
270
300 ***LOAD MACHINE CODE.
310 FOR ID=TH TO TH+13
320 READ VL;POKE ID,VL
330 NEXT
340
400 ***Z-80 BLOCK MOVE SUBROUTINE.
410 DATA 33,0,112 :LD HL,7000H (#28672D START VIDEO RAM)
420 DATA 17,1,112 :LD DE,7001H (#28673D NEXT OR DEST.)
430 DATA 1,255,7 :LD BC,07FFH (#2047D SIZE OF VIDEO RAM)
440 DATA 54,85 :LD (HL),55H (#85D YELLOW OR CHAR."U")
450 DATA 237,176 :LDIR (BLOCK MOVE INSTRUCTION)
460 DATA 201 :RET
470
500 ***INITIALIZE DELAYS - CONTROL SPEED OF EXECUTION BY D.
510 T=0 :***TONE 0 IS REST. RANGE IS 0 TO 31
520 D=4 :***DURATION 9 IS LONG. RANGE IS 1 TO 9
530 P=30744 :***ADDR. FOR INVERSE CONTROL.
540 POKE P,0 :***SET UP SCREEN.
550
600 ***SET UP DEMO LOOP.
610 FOR ID=0 TO 255
620 POKE AD,ID :***SET CHARACTER BYTE.
630 :***SCREEN MESSAGE.
640 MODE(0) :***SET A/G LD.
650 POKE P,0 :***SET INV LD.
660 PRINT#234," CHAR = ";ID;SOUND T,D
670 :***LO-RES SCREENS.
680 :***LO-RES GREEN CHARACTER ON BLACK BACKGROUND.
690 X=USR(0);COLOR,0;SOUND T,D;***SET CSS LD.
700 :***LO-RES ORANGE CHARACTER ON BLACK BACKGROUND.
710 COLOR,1;SOUND T,D :***SET CSS HI.
720 POKE P,1 :***SET INV HI.
730 :***LO-RES BLACK CHARACTER ON GREEN BACKGROUND.
740 X=USR(0);COLOR,0;SOUND T,D;***SET CSS LD.
750 :***LO-RES BLACK CHARACTER ON ORANGE BACKGROUND.
760 COLOR,1;SOUND T,D :***SET CSS HI.
770 :***HI-RES SCREENS.
780 MODE(1) :***SET A/G HI.
790 POKE P,0 :***SET INV LD.
800 :***HI-RES COLOR SET 0 - GREEN SURROUND.
810 X=USR(0);COLOR,0;SOUND T,D;***SET CSS LD.
820 :***HI-RES COLOR SET 1 - BUFF SURROUND.
830 COLOR,1;SOUND T,D :***SET CSS HI.
840 POKE P,1 :***SET INV HI.
850 :***HI-RES COLOR SET 0.
860 X=USR(0);COLOR,0;SOUND T,D;***SET CSS LD.
870 :***HI-RES COLOR SET 1.
880 COLOR,1;SOUND T,D :***SET CSS HI.
890 :***RESET CONTROLS.
900 POKE P,0;COLOR,0;CLS
910 NEXT
920
930 ***RESET TOM POINTERS.
940 POKE 30898,HI;POKE 30897,LI
950 STOP:END

```

**TABLE 3:**  
**CONFIGURATION OF BYTES IN MODE (1).**

3	2	1	0	Element #
0 0 0	0 0 0	0 0 0	0 0 0	Bin. = four GREEN/BUFF elements. Dec. = 00
0 1 64	0 1 16	0 1 4	0 1 1	Bin. = four YELLOW/CYAN elements. Dec. = 85D
1 0 128	1 0 32	1 0 8	1 0 2	Bin. = four BLUE/MAGENTA elements. Dec. = 170D
1 1 192	1 1 48	1 1 12	1 1 3	Bin. = four RED/ORANGE elements. Dec. = 255D

The decimal numbers corresponding to each element position AND colour provide the value that needs to be POKE'd or loaded.

of the screen. If, however, a striped screen consisting of RED-GREEN-BLUE-YELLOW vertical bands is required, then POKE (192 + 0 + 8 + 1) or 201D.

Although only four colours are available, there are two colour sets available. These are called by the COLOR command.

COLOR, 0 sets the background colour to green and the 'strong' colours of yellow, blue and red are available.

COLOR, 1 sets the background to buff and the 'pastelle' colours of cyan, magenta and orange are available.

To think back to the RESET command mentioned before, it should be apparent that this command simply resets each dibit or element back to 00B, or the background colour.

## Finale

Well there we have it! For those who have perservered thus far I have included Listing 4 which is entitled '2000 VZ Screens'. It is about as exciting as watching a Late Night Movie — and takes about as long to run! Actually it illustrates all of the features discussed in this article. For those who wish to sit-it-out — watch those control lines operate!

AEM Oct. 86 p. 121.

4 of 4.

## Addendum.

Add the following lines to listing 4.

145 CLEAR 50

945 CLEAR 50

A quicker way is to POKE values into each byte, thereby setting four elements at a time. Listing 2 demonstrates this technique. This program also fills the entire hi-res screen with elements whose colours are determined by the variables V%.

The quickest way is to use a machine language program to load appropriate values into the video RAM. This technique is a very rapid way to fill the screen. Listing 3 is an example of this method. This program POKes machine code into hi-memory. The subroutine uses the very efficient Z80 Block Move command to fill the screen according to the value stored at address -28677D. It is fast!

Both of the last two methods require that an understanding of the value to enter into RAM is known. This requires a knowledge of how each byte is organised in CG2 mode.

As mentioned previously, each byte controls four elements which can be selected from four colours. Bits are treated in pairs (dibits!) with each pair corresponding to an element. Each dibit can have a value of 00B to 11B to indicate colour. This is set out on Table 3.

Four example, suppose we want an entirely BLUE screen. Then POKE (128 + 32 + 8 + 2) or 170D into the appropriate area